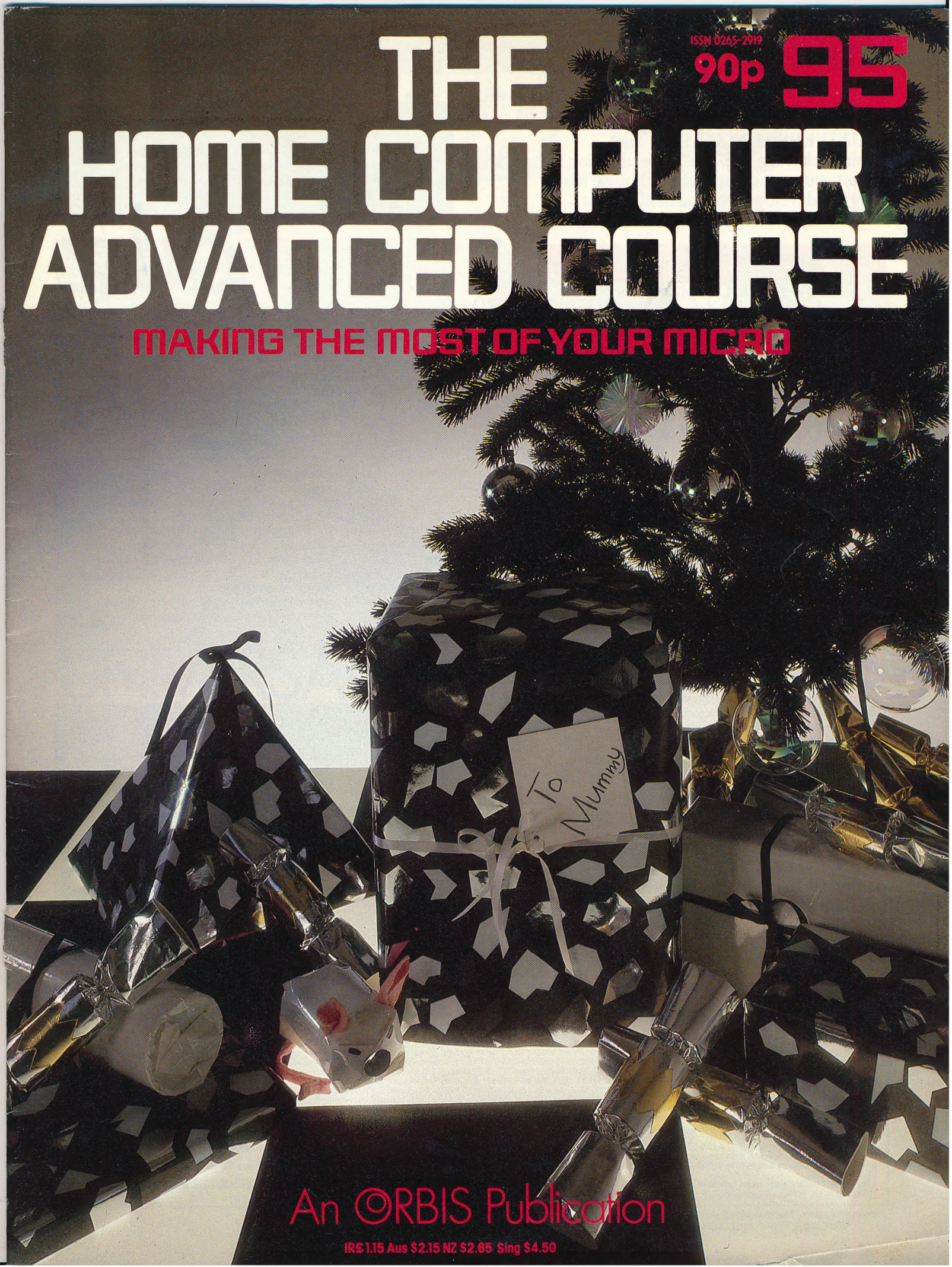
ISSN 0265-2919

90p 95

# THE HOME COMPUTER ADVANCED COURSE

## MAKING THE MOST OF YOUR MICRO




An ORBIS Publication

IR£1.15 Aus $2.15 NZ $2.65 Sing $4.50

# CONTENTS


APPLICATION

**A UNIVERSAL MACHINE** The Dynabook was a late-1960s specification for a multi-purpose computer. Today, we have only just developed the technology that could make such a machine possible — 1881

HARDWARE

**IN THE FUTURE** Our prediction for the average home micro of the early 1990s — 1889

SOFTWARE

**A WAY IN** We look at Unix's larger scale facilities and explore the 'ingres' database — 1886

**HARD COPY** A range of computer-related books for further reading — 1894

**SMOOTHING THE TRANSITION** The Supersoft Mikro is an assembler for the Commodore 64 — 1900

COMPUTER SCIENCE

**LANGUAGE ASSESSMENT** Some of the factors to weigh when choosing a language — 1892

JARGON

**WIMP TO ZERO PAGE** Our final glossary of computing terms — 1888

PROGRAMMING PROJECTS

**EXPANDING ON THE TOPIC** We complete our text compression program — 1883

MACHINE CODE

**IN TRANSLATION** A discussion of the part played by the assembler in translating assembly language instructions into 68000 machine code — 1896


## Next Week

**• In the final instalment, we present a complete index to the eight volumes of the course. Issue 96 will also include a series of ready-reference guides to the main subjects covered in each sub-section.**

## QUIZ

1) Which eight-bit processor allows the programmer to set zero page anywhere in memory?
2) What are assembler directives?
3) What is a 'smart card'?
4) Decode this 6502 format message:

10 03 BB 50 70 4E F1 FF C4 4F B0 29 F9 6E 5E 97 20

**Answers To Last Week's Quiz**

**1)** The Unibus is used to communicate with peripheral devices which have a non-critical access time. Peripheral devices that do need a very fast access time are connected to the Massbus.
**2)** Pseudo-ops are not part of the processor's instruction set and are used to instruct the assembler to perform a specific task.
**3)** Because of the very high packing density of data on a Winchester disk, it has to be sealed in an airtight container. This prevents dust entering and corrupting the data.
**4)** C contains a number of memory allocation commands so that the programs can be reconfigured to work on a variety of machines.

## Erratum

In our recent Pontoon programming project the following lines were omitted:

**Spectrum:**

```
610 LET P(1)=1: LET P(2)=1
1325 LET D(PL,P(PL),1)=CN:LET
D(PL,P,(PL),2)=SU:LET P(PL)=P(PL)+1
Add RETURN to the end of line 680
```

**Amstrad, Commodore 64 and BBC Micro:**

```
610 HP(1)=1:HP(2)=1
1325 HD(PL,HP(PL),1)=CN:HD(PL,HP(PL),2)
=SU:HP(PL)=HP(PL)+1
```


**Editor** Stephen Cooke; **Art Editor** Claudia Zeff; **Deputy Editor** Steve Colwill; **Production Editor** Bobby Pickering; **Designer** Julian Dorr; **Staff Writer** Steve Malone; **Art Assistants** Caroline Clayton; **Sub Editor** Jon Kaye; **Contributors** Mike Curtis, Steve Malone, David Fensome, Dick Pountain, Steve Cooke, Steve Colwill, Max Phillips; **Software Consultants** Pilot Software City; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd: Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Maurice Geller; **Production Assistant** Susan Brown; **Subscription Manager** Christine Allen; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 14 Rathbone Place, London W1P 1DE; **© APSIF Copenhagen 1985; © Orbis Publishing Ltd 1985: Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Heanor Gate Printing Ltd, Derby**

**HOW TO OBTAIN ISSUES AND BINDERS FOR THE HOME COMPUTER ADVANCED COURSE** – Issues can be obtained by placing an order with your newsagent or direct from our subscription department. If you have any difficulty obtaining any back issues from your newsagent, please write to us stating the issue(s) required and enclosing a cheque for the cover price of the issue(s). **AUSTRALIA** – please write to: Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G, Melbourne, Victoria 3001. **MALTA, NEW ZEALAND & SOUTH AFRICA –** Back numbers are available at cover price from your newsagent. In case of difficulty, write to the address given for binders.
**UK/EIRE** – Issue Price: 90p/IR£1.15. Subscription: 6 months: £26.00. 1 Year: £52.00. Binder: please send £3.95 per binder, or take advantage of our special offer in early issues. **EUROPE** – Issue Price: 90p. Subscription: 6 months air: £44.72. Surface: £36.14. 1 year air: £89.44. Surface: £72.28. Binder: £5.00. Airmail: £8.25. **MALTA** – Obtain binders from your newsagent or Miller (Malta) Ltd, MA Vassalli Street, Valetta, Malta. Price: £3.95. **MIDDLE EAST** – Issue Price: 90p. Subscription: 6 months air: £50.18. Surface: £36.14. 1 year air: £100.36. Surface: £72.28. Binder: £5.00. Airmail: £8.25. **AMERICAS/ASIA/AFRICA** – Issue Price: US/CAN$1.95/90p. Subscription: 6 months air: £59.54. Surface: £36.14. 1 year air: £119.08. Surface: £72.28. Binder: £5.00. Airmail: £9.50. **SOUTH AFRICA** – Obtain binders from any branch of Central News Agency or Intermag, PO Box 57394, Springfield 2137. **SINGAPORE** – Issue Price: Sing $4.50. Obtain binders from MPH Distributors, 601 Sims Drive, 03-07-21, Singapore 1438. **AUSTRALASIA/FAR EAST** – Issue Price: 90p. Subscription: 6 months air: £64.22. Surface: £36.14. 1 year air £128.44. Surface: £72.28. Binder: £5.00. Airmail: £9.75. **AUSTRALIA** – Issue Price: AusS2.15. Obtain binders from First Post Pty Ltd, 23 Chandos Street, St Leonards, NSW 2065. **NEW ZEALAND** – Issue Price: NZS2.65. Obtain binders from your newsagent or Gordon & Gotch (NZ) Ltd, PO Box 1595, Wellington.
**ADDRESS FOR BINDERS AND BACK ISSUES** – Orbis Publishing Limited, Orbis House, Bedfordbury, London WC2 4BT. Telephone 01-379 5211. Cheques/postal orders should be made payable to Orbis Publishing Limited. Binder prices include postage and packing and prices are in stirling. Back issues are sold at the cover price, and we do not charge carriage in the UK.
**NOTE** – Binders and back issues are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK and Australian markets only. Binders and Issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.
**ADDRESS FOR SUBSCRIPTIONS** – Orbis Publishing Limited, Hurst Farm, Baydon Road, Lambourn Woodlands, Newbury Berks, RG16 7TW. Telephone: 0488-72666. All cheques/postal orders should be made payable to Orbis Publishing Limited, Postage and packaging is included in subscription rates, and prices are given in sterling.

COVER PHOTOGRAPHY BY MARCUS WILSON-SMITH

KEVIN JONES ASSOCIATES

## A Dynamic Future

Originally envisaged by Alan Kay of Xerox, the futuristic Dynabook could be built using today's technology — although the price involved would make it an impractical proposition. In the future, however, a device like the one illustrated could become an essential aid for everyone — especially those groups (the aged, for example) who are becoming increasingly isolated in a fast-moving technological society.

A redefinable touch-sensitive screen for input, a high-resolution LCD display, voice input and cellular radio networking would enable the user to control and interact with systems ranging from local weather reports through to personal banking services. Since the service would be self-financing, the standard unit could be supplied either free of charge or for nominal rental by central government

# A UNIVERSAL MACHINE

**The Dynabook is a specification for a universally useful and portable computer, first put forward in 1969 before the invention of the microprocessor. We look at the technology that would be required to produce such a machine, and consider the problems that must still be overcome before it is realised.**

In 1969, a young American computer science student named Alan Kay presented his PhD thesis, in which he imagined the invention of a universally useful portable computer called the Dynabook. The Dynabook was to provide for all a user's information processing needs, including audio and visual communications and access to libraries of public information. In 1969, the microprocessor hadn't been invented and computers ranged in size from a large fridge to several wardrobes; what's more, their price tags ended in at least five zeros. Kay's Dynabook was very much a dream of the future.

Kay went on in 1971 to work at Xerox's Palo Alto Research Centre (PARC) and was influential in the Language Research Group that devised SMALLTALK, which was originally conceived as the Dynabook's programming language and operating system. The Xerox team never built the Dynabook — even after the invention of the microprocessor, in 1971, the available technology just wasn't powerful enough. (They did, however, build a suitcase-sized portable computer some five years before Osborne reinvented the idea.) Nevertheless, the ideas contained in SMALLTALK, such as windows, icons and mice, slowly diffused into the computer industry, and eventually led to Apple's Lisa and Macintosh, and to the Atari 520ST. Alan Kay himself now works as a research fellow at the Apple Corporation.

Such 'ancient' history is instructive because the time for building the Dynabook is almost upon us — the necessary components are either available now or soon will be. Today, Alan Kay's vision has begun to look realisable.

The original concept of the Dynabook was that it would be a device about the size of a hardback book (hence the name), made fully portable

## Technological Requirements

This is, of course, a very tall order. But let's take a look at some of the existing or emerging technologies that could make it happen.

- **Processing Power:** At the heart of the machine there needs to be a lot of processing power. 32-bit microprocessors with the power of a mainframe are now available as single chips off the shelf. The list, which is growing by the month, includes the Motorola 68020, the Inmos T414 Transputer, the Intel 80386, and the Acorn ARM. Each of these is capable of executing between three and 10 million instructions per second.

- **Memory:** All the major memory manufacturers are now producing 256 Kbit RAM chips, with many personal computers on the market incorporating them. But the manufacturers are all working hard on the next generation of one-megabit chips, and a few (including AT&T) are actually advertising sample parts.

- **Display:** Several manufacturers are currently producing flat monochrome liquid crystal displays that can emulate an IBM PC's 640 by 200 graphics screen. Electro-luminescent panels are now being used to backlight such displays and overcome their major shortcoming — namely, their low-contrast and limited viewing angles. Moreover, flat display technology is developing faster than just about any other sector of the industry, with promising new developments announced almost every week. Japanese manufacturers, including Epson, have already shown prototype colour LCDs, while Toshiba and others are working on self-luminous plasma tubes. The impetus for most of this research work is the quest for a viable flat-screen television rather than a computer display, which is precisely what the Dynabook needs.

  Just around the corner too are touch-sensitive flat screen displays that are likely to do away with the keyboard. A large display on the surface of the computer would include a touch-sensitive 'picture' of a keyboard, which can be customised under software control. At each stage of a program, you would see a different set of special labelled 'keys', and pictures might be drawn directly with a stylus rather than a mouse.

- **Power Source:** Providing electrical power is actually the biggest problem. The requirements of the new displays and large memories would severely tax the lithium batteries used in present day lap-held portables, and battery technology is not necessarily progressing at the same pace as the development of semiconductors. However, new types of batteries with higher energy densities are slowly emerging. And, at the same time, the power requirements of chips are declining as more manufacturers turn to low-consumption CMOS technology.

- **Mass-Storage:** The compact laser disc promises to make all forms of magnetic storage, such as the tape, floppy and Winchester disk, redundant. The Dynabook could, for example, use read-only compact discs as a means of distribution for 'books' of too specialised interest to be put onto public networks, as well as music and video publications.

  Another form of mass storage that may prove important is the 'smart card'. This is a credit card-sized memory device that can be used for anything from software distribution to keeping bank accounts — the UK company Cumana now distributes home computer software with its Astron card. In the near future, such cards might be used for paying for goods and services via the Dynabook.

- **Sound:** It has been possible for some time to get all the components of an audio system on a single chip, a good example of which is the chip contained in the Sony Walkman. Radio and television receivers are equally affected (hence pocket television sets) and music synthesiser chips are progressing apace.

  Voice, however, is a different matter. Speech synthesis is fairly easy to achieve, and modern music synthesiser chips can perform it quite well, but natural language understanding is still at an early stage of development.

- **Communications:** The remaining pieces of the Dynabook jigsaw lie in the area of communications. Telephone modems on a chip are in common use now, but they are not good enough for through battery operation. It would include a colour graphics/television display and be capable of presenting and processing text, pictures and sound. Above all, it would be a powerful communications medium.

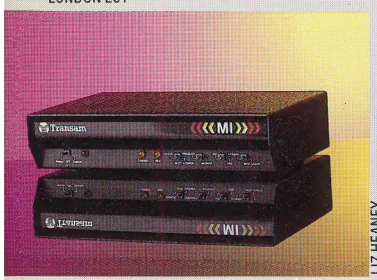
A truly revolutionary computer must be not only portable, but able to exchange all kinds of data with other users, providing us with a predominant means of long-distance communication, as the telephone is now. In short, we should be able to transmit a letter, complete with pictures and sound, and receive the same.

But this is only the start. We should have access to a variety of public services — from libraries, which would provide us with books and computer programs that could be downloaded onto the system, as well as news, weather, entertainment and educational facilities. Inspecting bank accounts, checking street maps, ordering goods and booking travel tickets should all be available — from any location and independent of telephone lines or power sockets.

Although much of the technology described under Technological Requirements is at the leading edge of development, and hence expensive, financing the Dynabook is really not the problem — if properly realised it would be the ultimate mass production item (insofar as everyone in the world could have one). With production volumes like that, unit hardware cost would hardly be a consideration. The marketing problems that matter so much to present day personal computer manufacturers would become quite trivial if computers were actually *useful* to people — and the Dynabook would be not so much useful as indispensable.

Nor does the biggest problem lie in the technology. It is, rather, in finding the incentive to put such a system together. Ultimately, it will be a political problem, since only governments have the resources needed for such an undertaking. Yet everything the Dynabook would do is already done, often at state expense, by scores of different, inefficient and expensive systems. It's not too far-fetched to imagine a situation in which the state could afford to *give away* Dynabooks, just as the French government is currently doing with microcomputer telephone terminals (they are cheaper than printing and distributing paper telephone directories). The services rather than the hardware would be paid for, just as we now pay phone bills and TV licences.

COURTESY OF TRANSAM, 59-61 THEOBALDS ROAD, LONDON EC1




LIZ HEANEY

**On The Air**
The introduction of a cellular radio network in the mid-1980s has spurred the development of cellular radio modems. The model shown, the Transam M1, is an intelligent modem offering auto-answer and auto-dial facilities if used with a standard BT telephone. However, it also features a 15-way D-type connector to hook up to a transceiver, enabling data to be sent and received via the cellular radio network Dynabook. For complete freedom of movement, the link must be achieved by radio broadcast, not by cable. Significantly, the first cellular radio modem chips are just beginning to appear.

For the Dynabook concept to be fully realised, we must envisage a multi-layered, world-wide communications network. Large mainframe computers contain the basic information, and the first step in reaching them is a cellular radio link to a local computer station. This network node communicates with regional centres by ground-based fibreoptic cables, and these, in turn, communicate by more optic fibres and satellite links to the next levels in the hierarchy. Many of these components are already being installed; what's preventing any further development is a lack of standardisation.

# EXPANDING ON THE TOPIC

**In this final instalment of our series on data compression techniques, we provide the 6502 Text Expansion Listing and a BASIC Loader program. We also present a Driver program for both Z80 and 6502 versions, which allows you to implement these techniques from BASIC.**

6502 users who do not have an assembler, or wish to save entry time, can POKE the text compression/expansion routines into memory using the 6502 BASIC Loader program. Once this program has been RUN, it may be deleted and replaced with the BASIC Driver program. The latter is suitable for use with both the 6502 routines and the Z80 versions previously published.

The BASIC Driver program asks you to input the string to be compressed and to specify an address for the compressed data to be stored. You will, of course, need to note this address should you wish to retrieve the data at some later stage. This is done simply by specifying the address of the compressed data so that it may be accessed by the decompression routine.

The Z80 compression/decompression routine is at location 30000, which should suit most micros. If you have an assembler you can relocate the code simply by changing the ORG statement. The BASIC Driver program can be modified for inclusion in your own programs, perhaps to provide facilities for building up larger files of compressed text composed of 255-byte records.

## 6502 Text Expansion

The assembly language routine following should be added to the 6502 compression routine given on page 1866, inserting it before the final data tables. This routine has been written on the Commodore 64. BBC Micro users can convert it by making the following alterations:

- Change the zero page pointer allocations, ZPTR1 to ZPTR4, to addresses &80 to &87.
- Command format LABEL *=*+2 should be changed to .LABEL EQUW
- Command format LABEL *=*+1 should be changed to .LABEL EQUB
- Command format .BYT should be changed to EQUS
- Call addresses will depend on where the routine is assembled. The labels START and EXPAND denote the call addresses of the compression and expansion routines and can be accessed from BASIC

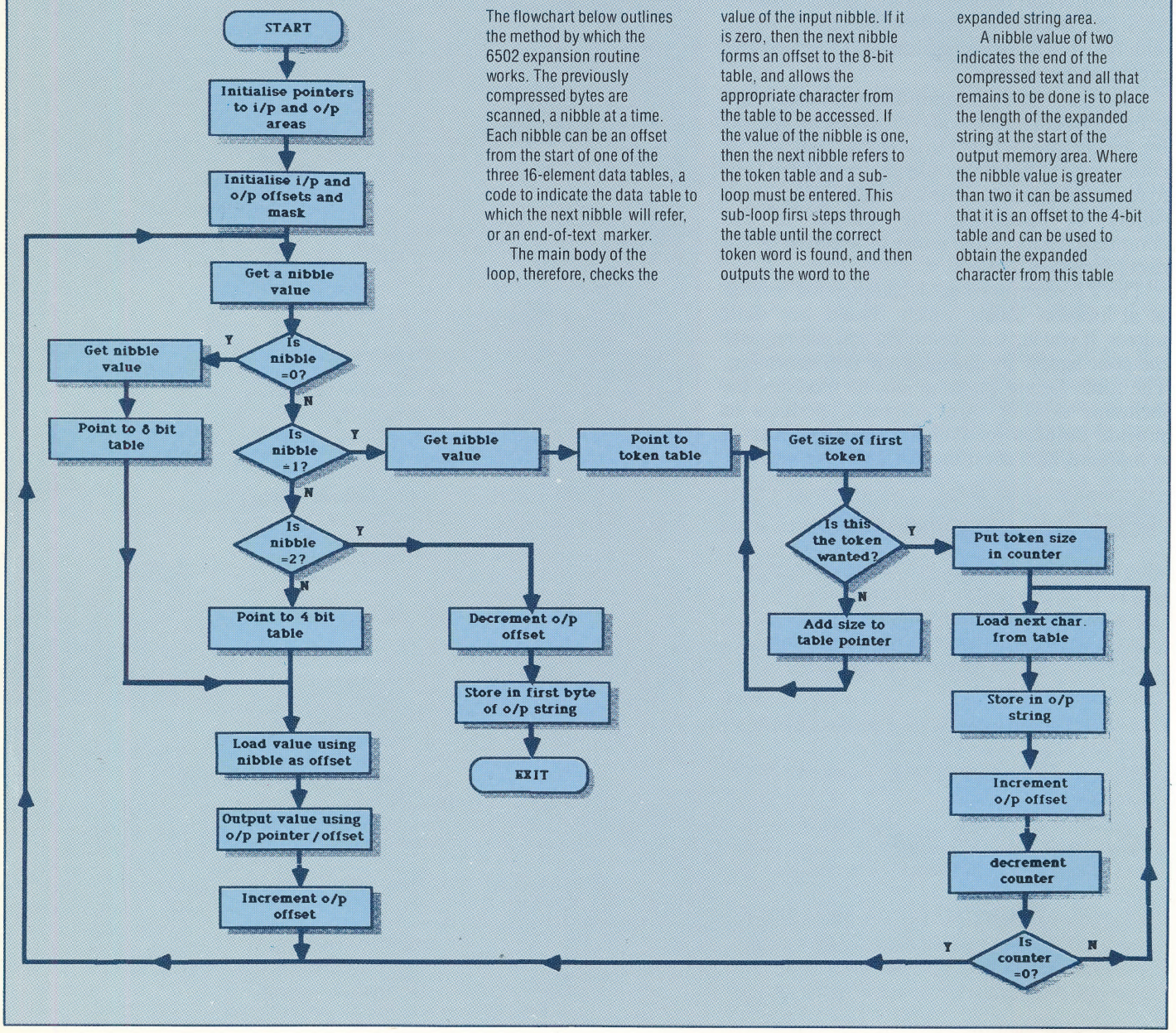
```
;
;++++ 6502 EXPAND ROUTINE +++++
EXPAND LDA INPUT
       STA ZPTR1
       LDA INPUT+1        ;SET UP 0 PAGE
       STA ZPTR1+1        ;PTRS TO POINT
       LDA OUTPUT         ;AT INPUT AND
       STA ZPTR2          ;OUTPUT AREAS
       LDA OUTPUT+1
       STA ZPTR2+1

       LDA #255           ;INITIALISE
       STA MASK           ;OFFSETS
       LDA #1             ;AND MASK
       STA LEN            ;INIT I/P OFFSET
       STA OUTOFF         ;INIT O/P OFFSET
;++ MAIN LOOP STARTS HERE ++
NEXNIB JSR GETNIB
       BEQ OUT8BT         ;0 NIBBLE MEANS 8BIT CODE
       CMP #2
       BEQ EXIT           ;IS IT END MARKER?
       BCC OUTTOK         ;IS IT TOKEN?
;++HANDLE 4BIT CODE ++
       TAY                ;NIBBLE VALUE INTO Y
       LDA #<TAB4BT
       STA ZPTR3          ;SET ZPTR3 TO
       LDA #>TAB4BT       ;START OF 4 BIT
       STA ZPTR3+1        ;TABLE
OUTCHR LDA (ZPTR3),Y      ;GET CHAR FROM TABLE
       LDY OUTOFF         ;SET UP Y FOR O/P
       STA (ZPTR2),Y      ;STORE IN O/P STRING
       INC OUTOFF         ;BUMP O/P OFFSET POINTER
       JMP NEXNIB         ;GO FOR NEXT NIBBLE
;++ HANDLE 8BIT CODE ++
OUT8BT JSR GETNIB         ;GET CODE NIBBLE
       TAY                ;PUT IT IN Y
       LDA #<TAB8BT       ;SET ZPTR3 TO
       STA ZPTR3          ;POINT AT START
       LDA #>TAB8BT       ;OF 8 BIT TABLE
       STA ZPTR3+1
       JMP OUTCHR         ;OUTPUT IT
;++ HANDLE TOKEN ++
OUTTOK JSR GETNIB         ;GET TOKEN PTR
       STA TABCNT         ;STORE IT AWAY
       LDA #<TOKTAB       ;SET ZPTR3 TO
       STA ZPTR3          ;LOOK AT TOKEN
       LDA #>TOKTAB       ;TABLE START
       STA ZPTR3+1
       LDX #0
       LDY #0
TOKA   LDA (ZPTR3),Y      ;GET LENGTH OF TOKEN
       CPX TABCNT         ;IF WE'VE GOT THE
       BEQ FOUND          ;ONE WE WANT -BRANCH
       CLC
       ADC #1
       ADC ZPTR3          ;SET ZPTR3 TO START
       STA ZPTR3          ;OF NEXT TOKEN
       LDA ZPTR3+1
       ADC #0
       STA ZPTR3+1
       INX
       BNE TOKA           ;THIS SHOULD ALWAYS BRANCH!
FOUND  TAX                ;PUT TOKEN LENGTH IN X
       LDY #1             ;INITIALISE Y
FOUND1 LDA (ZPTR3),Y      ;GET TOKEN CHAR
       STY TEMP           ;TEMP STORE Y
       LDY OUTOFF         ;GET O/P OFFSET
       STA (ZPTR2),Y      ;STORE CHAR IN O/P STRING
       INC OUTOFF         ;BUMP O/P OFFSET
       LDY TEMP           ;GET TOKEN OFFSET BACK
       INY
       DEX
       BNE FOUND1         ;IF NOT END, GET NEXT CHAR
       JMP NEXNIB         ;REJOIN MAIN LOOP
;++ HANDLE EXIT ++
EXIT   LDA OUTOFF         ;GET O/P OFFSET
       SEC
       SBC #1             ;REDUCE BY ONE
       LDY #0             ;AND STORE AT START
```

```
        STA (ZPTR2),Y   ;OF O/P STRING
        RTS             ;RETURN TO SYSTEM
;++ GET A NIBBLE SUBROUTINE ++
GETNIB  LDA MASK
        BNE RIGHT       ;IS MASK 255?
        LDA #255        ;SET UP MASK
        STA MASK        ;FOR NEXT TIME
        LDY LEN         ;GET I/P OFFSET
        LDA (ZPTR1),Y   ;GET I/P BYTE
        AND #15         ;MASK IT
        INC LEN         ;BUMP I/P OFFSET
        RTS
RIGHT   LDA #0          ;SET UP MASK
        STA MASK        ;FOR NEXT TIME
        LDY LEN         ;GET I/P OFFSET
        LDA (ZPTR1),Y   ;GET I/P BYTE
        LSR A           ;MOVE LEFT NIBBLE
        LSR A           ;DOWN INTO RIGHT
        LSR A           ;HALF OF BYTE
        LSR A
        AND #15         ;MASK IT
        RTS
```

## Z80/6502 BASIC Driver

```
100 REM ********************************
110 REM ** Program to compress/expand **
120 REM **                            **
130 REM * a 255-byte string (U/C only) *
140 REM ********************************
150 CLS
160 INPUT "6502 version (Y/N)";a$
170 IF a$="y" OR a$="Y" THEN cf=1
180 INPUT "Expand or Compress (E/C)?";a$
190 IF a$="" THEN GOTO 180
200 IF a$="e" OR a$="E" THEN ec=1:GOTO 2
60
210 IF a$="c" OR a$="C" THEN ec=0:GOTO 4
30
```

## Expanding Out




The flowchart below outlines the method by which the 6502 expansion routine works. The previously compressed bytes are scanned, a nibble at a time. Each nibble can be an offset from the start of one of the three 16-element data tables, a code to indicate the data table to which the next nibble will refer, or an end-of-text marker.

The main body of the loop, therefore, checks the value of the input nibble. If it is zero, then the next nibble forms an offset to the 8-bit table, and allows the appropriate character from the table to be accessed. If the value of the nibble is one, then the next nibble refers to the token table and a sub-loop must be entered. This sub-loop first steps through the table until the correct token word is found, and then outputs the word to the expanded string area.

A nibble value of two indicates the end of the compressed text and all that remains to be done is to place the length of the expanded string at the start of the output memory area. Where the nibble value is greater than two it can be assumed that it is an offset to the 4-bit table and can be used to obtain the expanded character from this table

CAROLINE CLAYTON

```
220 GOTO 180
230 REM
240 REM ***Expand a compressed string***
250 REM
260 PRINT "Where is the string to expand
?"
270 INPUT i
280 PRINT "Where do you want the output?
"
290 REM Be careful not to overwrite mach
ine code!
300 INPUT o
310 IF cf=0 THEN GOSUB 660:REM z80 versi
on
320 IF cf=1 THEN GOSUB 700:REM 6502 vers
ion
330 FOR x=1 TO PEEK(o)
340 PRINT CHR$(PEEK(o+x));
350 NEXT x
360 PRINT; "Continue (y/n)? ": INPUT a$
370 IF a$()"Y" AND a$()"y" THEN STOP
380 PRINT: GOTO 180
390 STOP
400 REM
410 REM ****** Compress a string ******
420 REM
430 PRINT "What shall I compress? Rememb
er that"
440 PRINT "the routine accepts upper cas
e letters only."
450 PRINT
460 INPUT "String to compress: ";a$
470 PRINT "Where shall I put it?"
480 REM See warning in line 290!
490 INPUT i
500 PRINT "Where do you want the output?
"
510 INPUT o
520 POKE i,LEN(a$)
530 FOR x=1 TO LEN(a$)
540 POKE i+x,ASC(MID$(a$,x,1))
550 REM poke i+x, code a$(x to x) on Spe
ctrum
560 NEXT x
570 GOSUB 660
580 PRINT "Output length ";PEEK (o)
590 PRINT "Compression achieved: ";100-P
EEK(o)/LEN(a$)*100;"%"
600 PRINT: "Continue (y/n)? ": INPUT a$
610 IF a$()"Y" AND a$()"y" THEN STOP
620 PRINT: GOTO 180
630 POKE 30003,INT(i/256)
640 POKE 30002,i-PEEK(30003)*256
650 POKE 30005,INT(o/256)
660 POKE 30004,o-PEEK(30005)*256
670 RETURN
680 IF ec=1 THEN CALL 30270:RETURN
690 CALL 30000:IF PEEK(30006)()0 THEN PR
INT "Compression Failed":STOP
700 REM 6502 Commodore version
710 POKE 49153, INT(0/256)
720 POKE 49152, i-PEEK
730 POKE 49155, INT(i/256)
740 POKE 49154, i-PEEK(49155)*256
750 IF ec=1 THEN sys 49163
760 sys 49518: IF PEEK(49156)()0 THEN PR
INT"compression failed":STOP
770 RETURN
```

## 6502 BASIC Loader

```
10 REM ********************************
15 REM ** COMMODORE 64 COMPRESSION   **
20 REM ** AND EXPANSION BASIC LOADER **
25 REM ********************************
30 FOR I=49163 TO 49807:READ A:POKEI,A
35 CC=CC+A:NEXT I
40 READ CS:IF CS()CC THEN PRINT"ERROR":STOP
100 DATA173,2,192,133,139,173,3,192
110 DATA133,140,173,0,192,133,141,173
120 DATA1,192,133,142,169,1,141,7,192
130 DATA160,0,177,139,141,6,192,169
140 DATA255,141,5,192,200,177,139,32
150 DATA39,193,208,54,32,129,192,240
160 DATA55,32,242,192,240,10,32,2,193
170 DATA72,169,0,32,65,193,104,32,65
180 DATA193,204,6,192,144,220,169,0
190 DATA141,4,192,169,2,32,65,193,173
200 DATA5,192,240,3,206,7,192,160,0
210 DATA173,7,192,145,141,96,169,255
220 DATA141,4,192,96,72,169,1,32,65
230 DATA193,104,32,65,193,76,79,192,72
240 DATA152,72,140,10,192,165,139,24
250 DATA109,10,192,133,251,165,140,105
260 DATA0,133,252,169,83,133,254,169
270 DATA194,133,255,169,0,141,8,192
280 DATA160,0,177,254,72,170,240,65
290 DATA165,254,24,105,1,133,254,165
300 DATA255,105,0,133,255,177,251,209
310 DATA254,208,22,200,202,208,246,104
320 DATA104,136,140,10,192,24,109,10
330 DATA192,168,104,173,8,192,162,0,96
340 DATA104,141,10,192,165,254,24,109
350 DATA10,192,133,254,165,255,105,0
360 DATA133,255,238,8,192,76,162,192
370 DATA104,104,168,104,162,255,96,72
380 DATA140,10,192,169,51,133,251,169
390 DATA194,133,252,104,76,15,193,72
400 DATA140,10,192,169,67,133,251,169
410 DATA194,133,252,104,160,0,209,251
420 DATA240,11,200,192,16,208,247,172
430 DATA10,192,162,255,96,152,172,10
440 DATA192,162,0,96,201,32,240,16,201
450 DATA44,240,12,201,46,240,8,201,65
460 DATA144,7,201,91,176,3,162,0,96
470 DATA162,255,96,72,140,10,192,172,7
480 DATA192,173,5,192,208,17,104,17
490 DATA141,145,141,238,7,192,172,10
500 DATA192,169,255,141,5,192,96,104
510 DATA10,10,10,10,145,141,172,10,192
520 DATA169,0,141,5,192,96,173,2,192
530 DATA133,139,173,3,192,133,140,173
540 DATA0,192,133,141,173,1,192,133
550 DATA142,169,255,141,5,192,169,1
560 DATA141,6,192,141,7,192,32,13,194
570 DATA240,28,201,2,240,106,144,37
580 DATA168,169,51,133,251,169,194,133
590 DATA252,177,251,172,7,192,145,141
600 DATA238,7,192,76,143,193,32,13,194
610 DATA168,169,67,133,251,169,194,133
620 DATA252,76,163,193,32,13,194,141,9
630 DATA192,169,83,133,251,169,194,133
640 DATA252,162,0,160,0,177,251,236,9
650 DATA192,240,16,24,105,1,101,251
660 DATA133,251,165,252,105,0,133,252
670 DATA232,208,233,170,160,1,177,251
680 DATA140,10,192,172,7,192,145,141
690 DATA238,7,192,172,10,192,200,202
700 DATA208,236,76,143,193,173,7,192
710 DATA56,233,1,160,0,145,141,0,173,5
720 DATA192,208,16,169,255,141,5,192
730 DATA172,6,192,177,139,41,15,238,6
740 DATA192,96,169,0,141,5,192,172,6
750 DATA192,177,139,74,74,74,74,41,15
760 DATA96,0,0,0,70,76,68,72,83,73,82
770 DATA78,79,65,84,69,32,67,77,85,71
780 DATA89,80,87,66,86,75,88,74,81,90
790 DATA44,46,3,84,72,69,4,84,72,73,83
800 DATA4,84,72,65,84,2,73,70,3,89,79
810 DATA85,2,77,69,3,87,65,83,2,72,69
820 DATA3,83,72,69,4,84,72,69,89,2,79
830 DATA70,2,73,84,2,73,83,3,70,79,82
840 DATA2,79,78,2,84,79,0,7
850 DATA76822:REM*CHECKSUM*
```

# A WAY IN

**In this final instalment in our series on Unix, we turn our attention to the larger scale facilities this operating environment has to offer. In particular, we focus on text editors, databases and, to begin with, electronic mail — an in-built feature of Unix.**

Electronic mail technology has been around for some time now, though its infiltration into the market-place has been inhibited to an extent by its relative unfriendliness, slow speed and unreliability. The electronic mail system in Unix, the first operating system to include one as standard, allows communication between users of the same machine, and can be extended to cover communications between any two Unix-based machines over a suitable link. We'll look at how the system works between users of the same machine; since the operation is identical for remote users, this should prove a suitable introduction for both cases.

The system is activated by the mail command, which opens your personal mailbox file and provides access to all correspondence sent to you. All incoming messages are identified by a number and the 'ID' of the sender. If copies were sent to other users, this too is indicated. New and unread messages are flagged (each is normally accompanied by a one-line heading to identify the subject matter) and can be read either individually or in the order in which they arrived by simply pressing the Return key.

The system is identified by the & prompt, which has a variety of commands concerning mail handling and system functions, such as changing the directory, and you can exit with either the x or q command. The former leaves all mail that hasn't

## Database Dialogue

```
Berkeley 4.2 Vax/Unix (infc3)
Type (Ctrl-D) to disconnect

login: com-mcc
Password:
You are a normal user (class 3)
Jobs: 6 Superiors: 2 Maximum: 21
Last login: Thu Nov 14 13:47:42 on ttyn04

You have mail.                  (sign-on tells you if mail awaits)

%mail
Mail version 2.18 5/19/83. Type ? for help.
"/usr/spool/mail/com-mcc": 1 message 1 new
>N 1 com-vjp Fri Nov 15 09:52 11/271 "Specimen mailing"
&                               (simply hit RETURN to get new mail)
Message 1:
From com-vjp Fri Nov 15 09:52:35 1985
From: com-vjp (Vicki)
To: com-daf, com-mcc
Subject: Specimen mailing       (message title)
Hello                           (message contents)

& ?                             (gives a list of commands)
  Mail    Commands
t <message list>       types messages
n                      goto and type next message
e <message list>       edit messages
f <message list>       give head lines of messages
d <message list>       delete messages
s <message list> file  appends messages to files
u <message list>       undelete messages
r <message list>       reply to messages
pre <message list>     make messages go back to /usr/mail
m <user list>          mail to specific users
q                      quit, saving unresolved messages in mbox
x                      quit, do not remove system mailbox
h                      print out active message headers
!                      shell escape
c [directory]          chdir to directory or home if none given
A <message list>       consists of integers, ranges of same, or user names
                       separated by spaces. If omitted, Mail uses the last
                       message typed.
A <user list>          consists of user names or distribution names
                       separated by spaces .
Distribution names are defined in .sendrc in your home directory.

% ingres demo           (now run the ingres database using demo
                        data)
INGRES version 7.10 (10/27/81) login
Fri Nov 15 10:28:42 1985
go
* print parts           (commands entered into workspace and
                        activated by \g)
*\g
Executing...

parts relation
| pnum  | pname              | colour    | weight | qoh    |
|-------------------------------------------------------|
|    10 | byte-soap          | clear     |      0 |    143 |
|     1 | central processor  | pink      |     10 |      1 |
|    11 | card reader        | grey      |    327 |      0 |
|     2 | memory             | grey      |     20 |     32 |
|    12 | card punch         | grey      |    427 |      0 |
|     3 | disk drive         | black     |    685 |      2 |
|    13 | paper tape reader  | black     |    107 |      0 |
|     4 | tape drive         | black     |    450 |      4 |
|    14 | paper tape punch   | black     |    147 |      0 |
|     5 | tapes              | grey      |      1 |    250 |
|     6 | line printer       | yellow    |    578 |      3 |
|-------------------------------------------------------|
continue
* \p                     (shows current contents of workspace)
print parts
```

been deleted in the mailbox so that it will reappear when the system is next activated; the latter will extract the messages you've read and put them into your directory under a special file called mbox. At this point, they are available for editing and the other functions usually associated with text file manipulation, but cannot be accessed directly by the mail system.

You can send mail from within the system with the m command, or from without by giving the mail command followed by the addressee's name (or names). The normal login IDs are used and the list of names can be as long as you like; each will receive a copy of your message.

When mail is distributed regularly to a number of users, you can save yourself from having to retype a list of names each time with the alias facility (see page 1864). A special file is used for these aliases and for setting other mail parameters, such as wanting to be informed when mail has arrived (using either .mailrc or .sendrc).

## THE INGRES DATABASE

Provided with, or at least available for, most Unix systems is the 'ingres' database. Although its user interface is less friendly than most, it is a very powerful relational database — far more powerful, in fact, than most familiar systems such as dBase II (see page 1152). But like most aspects of Unix, ingres can be customised to suit your requirements if necessary, an example of which is shown.

No system is complete without some form of text editor — Unix provides at least three as standard and others can be added optionally. The most basic of these is ed, a line editor similar to both ed on CP/M and edline on MS-DOS. As with all line editors, it's rather awkward to use, but it has the advantage of having all editing functions as commands. A file, therefore, can be set up that contains a sequence of commands, allowing the editing of a document to be carried out automatically by Unix. The two other editors are ded and vi, which can be used as either screen or line editors.

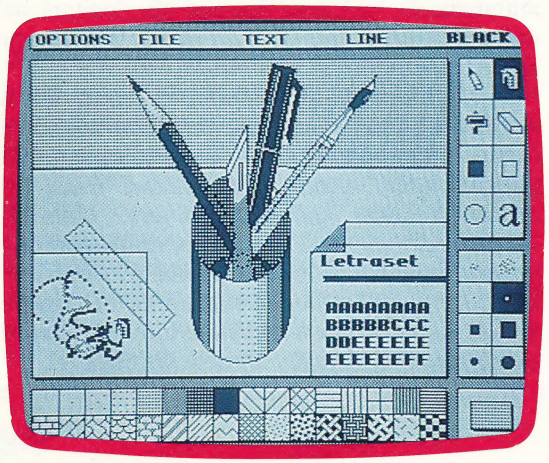
Finally, time has limited us to covering only a fraction of Unix's facilities, but those we have omitted concern mostly programming and program development, of interest more to advanced users than newcomers. Unix is undoubtedly one of the most important operating systems in terms of the development of modern computing, and although it may never achieve commercial success on micros, its influence can be clearly seen on virtually all new systems.

```
continue
* \r                          (clears workspace)
go
* range of p is parts         (specifies a particular part of database
                               to use, referred to as p)
* retrieve (p.pname)          (search for and display given field)
* \g
Executing...

| pname                    |
|--------------------------|
| byte-soap                |
| central processor        |
| card reader              |
| memory                   |
| card punch               |
| disk drive               |
| paper tape reader        |
| tape drive               |
| paper tape punch         |
| tapes                    |
| line printer             |
|--------------------------|
(11 tuples)

continue
* retrieve (p.pname,p.coclour)  (more than one field)
* where p.colour = "grey"       (specify search criterion)
* \g
Executing...


2100: line 1, Attribute 'coclour' not in relation parts
(this error message is not very helpful -we misspelt 'colour')
continue
* retrieve (p.pname,p.colour)
* where p.colour = "grey"
* \g
Executing...

| pname            | colour   |
|------------------------------|
| card reader      | grey     |
| memory           | grey     |
| card punch       | grey     |
| tapes            | grey     |
|------------------------------|
(4 tuples)

continue
* retrieve (p.pname,p.pnum,total=p.qoh * p.weight)
(values displayed can be calculated from fields and given a title)
* \g
Executing...

| pname            | pnum |  total |
|-----------------------------------|
| byte-soap        |   10 |      0 |
| central processor|    1 |     10 |
| card reader      |   11 |      0 |
| memory           |    2 |    640 |
| card punch       |   12 |      0 |
| disk drive       |    3 |   1370 |
| paper tape reader|   13 |      0 |
| tape drive       |    4 |   1800 |
| paper tape punch |   14 |      0 |
| tapes            |    5 |    250 |
| line printer     |    6 |   1734 |
|-----------------------------------|
(11 tuples)

continue
* \q                                  (quit ingres)
INGRES version 7.10 (10/27/81) logout
Fri Nov 15 10:41:03 1985
goodbye com-mcc -- come again
% logout
```

## WIMP

An acronym for 'windows icons mouse program', the concept behind a *WIMP* system is quite simple. Instead of a standard operating system in which the user types in a series of specialised commands to perform operations, processes are selected by positioning the cursor, with a hand-held mouse, over the relevant icon — a small pictorial representation of the task or operation.

The attraction of WIMPs is their considerable user-friendliness. It is far easier to move a file icon to the wastepaper bin icon, for example, than having to learn a series of instructions in order to accomplish the same task.




IAN McKINNELL

**Open Your Window**
WIMPs first made a big impression with the release of the Apple Lisa, and achieved widespread popularity on the Macintosh. It is likely that in a few years all new machines will be fitted with a WIMP system as standard

The WIMP system grew out of research performed by the Xerox Corporation in the late 1970s. The development led to the SMALLTALK environment based around the concept of object orientated programming (see page 1670). Although WIMPs were greeted with a degree of scepticism when they first appeared on commercial micros, most new machines either have WIMP systems built in or have the capability of incorporating systems designed for them.

## WORD

A group of bits taken together to constitute a unit by the computer is referred to as a *word*. The number of bits that make up a word is generally known as the 'word length', which is normally eight, 16 or 32 bits (one, two or four bytes). The word length of a computer is usually determined by the architecture of the CPU within the computer being used.

## WORD PROCESSING

*Word processing* is a facility that allows documents to be written into a computer. Once the text has been written, the document can be edited and formatted before being sent to the printer where it will be output as hard copy. Most of the better word processors have facilities that go beyond the simple input and editing of text, however. Many support features such as automatic wordwrap and justification, which removes the problem of deciding whether or not a word will fit onto a line.

Word processors are one of the principal applications of home and business micros. Consequently, there is an enormous quantity available, ranging from dedicated machines costing thousands of pounds, to sophisticated business programs priced at a few hundred, down to simple text editors costing just a few pounds. (See our series on word processors beginning on page 1695.)

## WORKSPACE

An area of memory, the *workspace* is set aside for the temporary storage of data and variables while the program is running.

## ZERO FLAG

The *zero flag* is one of a number of single-bit flags within the status register of the CPU that indicate when a certain condition has been fulfilled. If the result of an operation produces a zero, the zero flag will be set to one; otherwise it will continue to have the value zero.

The most common use of the zero flag is in machine code programs where a comparison is being made or a decrementing loop counter is being tested. The zero condition can be tested by the processor, and program control transferred to another part of the program, by 'branch if equal to zero' and 'branch if not equal to zero' instructions.

## ZERO PAGE

Absolute addresses on an eight-bit micro are written in terms of a high order byte and a low order byte. The high order byte is used to refer to the 'page' number of the address, while the low order byte points to the location within the page (a page is a group of 256 consecutive bytes in memory). *Zero page*, then, refers to the first page in memory where the high order byte is zero.

When we are using zero page addressing, therefore, there is no need to include the high order byte, and a single byte address can be used. Processors having zero page addressing modes will assume that the program is referring to zero page and access the address accordingly. Addressing the zero page is consequently much faster than addressing other pages in memory. It's often used as the workspace for the computer's operating system.

The concept of zero page addressing is most evident in the 6502 processor. Because this chip lacks many of the registers available on other chips, such as the Z80, zero page is used as a storage area for data required for fast access by the processor. Zero page effectively provides the 6502 with 256 extra 'registers'.

Although zero page refers to the first 256 bytes of memory, some processors, such as the Motorola 6809, allow the programmer to set a number in the direct page register that specifies the page number, which enables faster access to the selected page.

# IN THE FUTURE

**The speed at which microcomputer technology has developed over recent years would not inspire most of us to feel confident about predicting what machines will be like in the near future. However, we can make a reasonable guess about where current trends are leading, and here is a machine review that we think is more than possible in the early 1990s.**

The early 1990s have seen a number of features previously available only on business micros trickle down to the home micro end of the market. User-friendly front-ends coupled with an increasing number of facilities available to the home enthusiast have made the modern computer easy to use and given it on-board facilities for a wide range of applications.

The Conquer Chestnut is the latest in a line of 'home workstations' produced for the enormous domestic micro market. Now that the full potential of the home computer is at last being realised (recent polls have indicated that the user base is now equivalent to that of hi-fi systems), the market is being 'demand led' rather than 'supply led' — that is, the manufacturers are catering to customers' needs rather than producing hi-tech gadgets they hope will find a market.

At around £500, the Chestnut includes the most popular features of the last five years or so. Based around the familiar 68000 processor, the package includes a computer with integral colour monitor, a single double-sided double-density $3\frac{1}{2}$ in disk drive, a 10 Mbyte mini-hard disk and a megabyte of RAM. Measuring a mere 300 by 240 by 120mm, the computer housing continues the recent trend towards smaller 'footprint' machines that are less obtrusive on the desk.

The keyboard and trackball/mouse are connected to the computer via fibre optic cables. The keyboard is of the standard QWERTY format with 10 function keys and a numeric keypad that can double as a cursor cluster. The typewriter keyboard itself contains several control keys, such as Ctrl and Alt, which allow additional functions to be programmed into the machine.

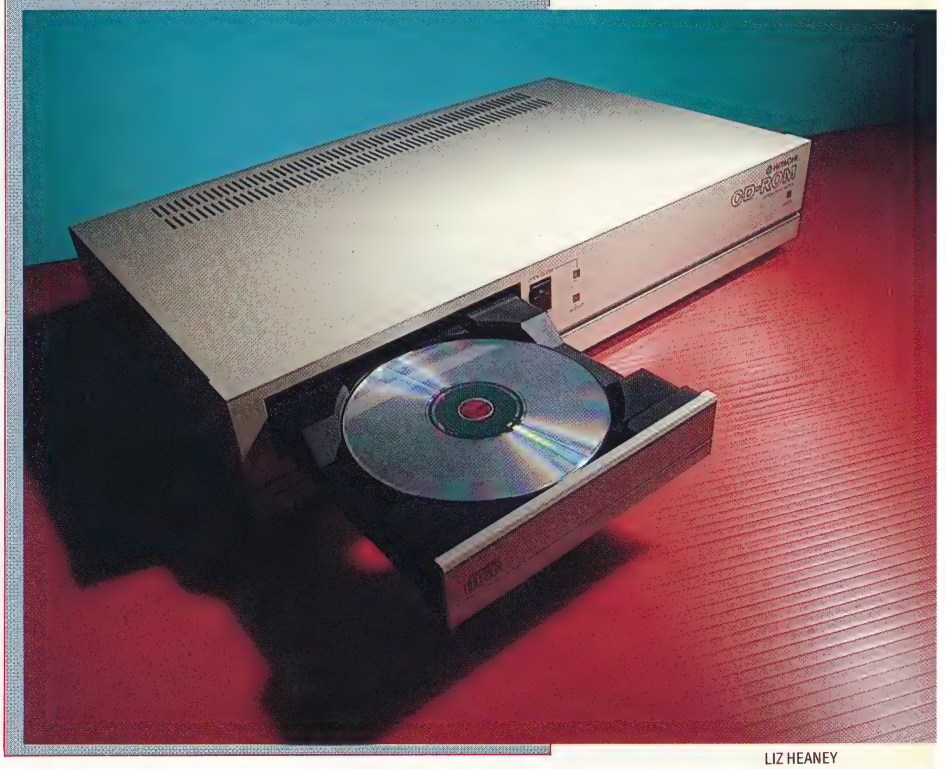
The Chestnut's size owes a great deal to the 'flat screen' technology that has made tremendous advances over the past few years. Not only is the monitor half the size of the cathode ray tube screens of the 1970s and 1980s, but there is no longer any distortion of images around the edges.

Underneath the monitor is the disk drive, which gives a single double-sided disk a capacity of one megabyte. A second drive can be fitted optionally in a space now covered with the Chestnut logo, just beneath the first drive (Conquer is intending to market a twin-drive model for business applications).

On the right-hand side of the main computer housing is the mouse controller port. There is also an expansion bus into which second processors, frame grabbers and further one-megabyte memory expansion modules can be fitted.

## Making Light Work

The CD-ROM drive is now considered an essential peripheral for anyone interested in microcomputers. Thus, in order to be competitive, the Chestnut has been provided with a suitable parallel port. This early Hitachi CD-ROM drive held 552 Mbytes of data and had a data transfer rate of 176 Kbytes per second




LIZ HEANEY

## COMMUNICATION PORTS

The rear panel of the machine houses the standard peripheral and communication ports. On the far right are a pair of hi-fi sockets that allow you to plug into a stereo system. This lets you take full advantage of the Chestnut's superb sound, provided by the Y/S2416 synthesiser chip that has eight oscillators on board and a range of five octaves. A full polyphonic sound can thus be produced with an electronic piano keyboard.

The user port, which accommodates music

keyboards or robotic devices, is next, followed by the serial MIDI sockets. Although most musicians are now using the MIDI-P (parallel) format, Conquer has decided to include with the Chestnut the old MIDI serial ports to take advantage of the large number of LAN packages produced for the MIDI-S standard.

Of course, it's no longer necessary to program the synthesiser chip directly. Because the MIDI software is now provided on-board, programming the chip is simply a matter of configuring the synthesiser as a MIDI device and sending it the appropriate codes.

Like most modern micros, all serial connections for the MIDI and on-board modem are contained on a single chip. Access to the telephone network is provided by a pair of BT telephone jacks,opening up for you the full range of available communications. These include Micronet, Final Frontier (the multi-user arcade game) and Telemarket, a service whereby games and applications programs can be purchased via the telephone network and downloaded onto the user's hard disk.

A feature of the new generation of home micros that has clearly captured the public's imagination has been the possibility of reliable speech recognition and synthesis. This has come about not from any technological innovation, but rather from the price of memory plummeting in recent years and the increased availability of high-performance processors. This has made available enough memory to store the voice templates that are required for the process.

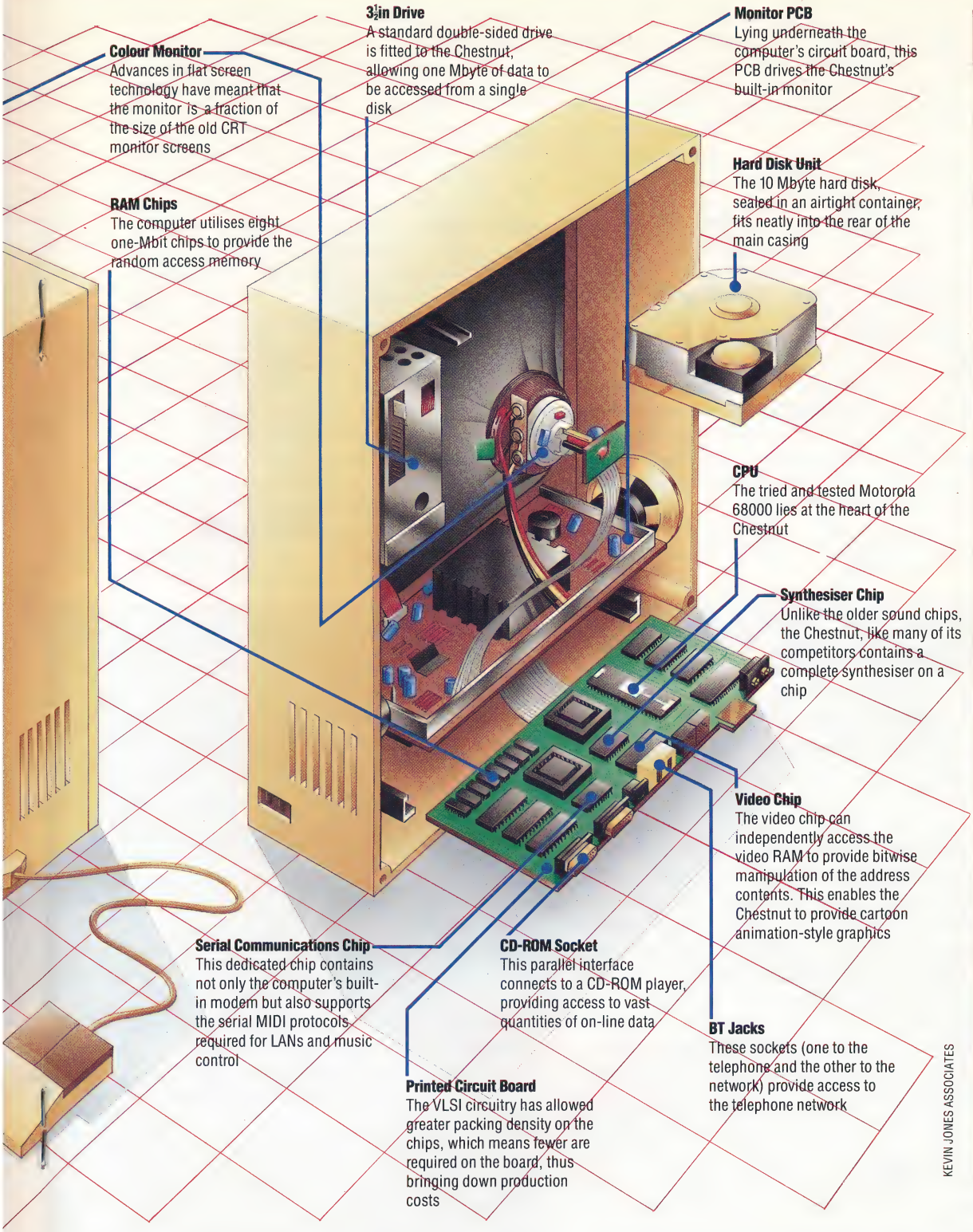
A microphone plugged into the D/A jack socket on the rear of the Chestnut will transmit spoken commands to the computer, which will search the templates to find a match and respond if successful. By using these same templates and the synthesiser chip, the Chestnut achieves a comprehensive level of voice synthesis and a limited conversational capability.

Next to the D/A interface is the customary Centronics printer port, and on the far left is a parallel interface for a CD-ROM drive. Conquer expects this to be a major selling point, since users will be able to access the large amount of database material, such as the Encyclopaedia Britannica, which has appeared recently on CD-ROM. Avid adventurers will be able to play the popular 500 Mbyte Odyssey, which takes full advantage of this technology — rapid access of massive storage has allowed the programmers to create an entirely voice-driven adventure with detailed cartoon graphics.

## MONITOR AND SOFTWARE

The high-resolution monitor included with the Chestnut is capable of displaying over 1,000 colours at any one time. The video blitter chip provides a smooth and realistic display in Animation mode and compares well with similarly priced systems on the market.

The WIMPOS-3 front-end icon-driven operating system is similar to previous versions developed by Conquer. The only difference seems to be that there is more scope for manipulating the windows and icons via spoken commands, although the review model did prove unreliable at times, particularly when it was spoken to at anything quicker than a drawl. For the time being at least, it would seem that using the mouse, although dated, is preferable to risking the loss of files through the machine's misinterpretation of vocalised instructions.




KEVIN JONES ASSOCIATES

## Conquer Chestnut

**PRICE**
£500 inc VAT

**DIMENSIONS**
300x240x120mm

**CPU**
68000 running at 8 MHz

**MEMORY**
1 Mbyte RAM; 128 Kbyte ROM

**SCREEN**
Text resolution 80x32; graphics resolution 640x256 pixels, allowing up to 1,000 colours to be displayed at any one time

**INTERFACES**
Expansion bus, mouse/trackball socket, stereo hi-fi sockets, user port, twin MIDI ports, twin BT sockets, A/D socket, parallel CD-ROM port

**LANGUAGES AVAILABLE**
BASIC, C compiler

**DISK DRIVE**
Single 3½in double-sided, double-density one Mbyte disk drive, with a second drive optional

**OPERATING SYSTEM**
WIMPOS-3

**DOCUMENTATION**
The documentation for the Chestnut is thorough and gives full details of the system. As a departure, Conquer is making the manuals available on CD-ROM, as well as on the normal hard copy version

**STRENGTHS**
The Chestnut offers a number of facilities that until recently have only been available on more expensive machines

**WEAKNESSES**
The machine offers nothing that is radically new and may have trouble making its voice heard among the 'leading edge' products on the market

The software bundled on the computer's hard disk includes the ChestWrite word processing program, the fully complemented ChestPaint, ChestTalk, for developing user-defined speech templates, as well as an expert system, ChestComplaint, which allows you to diagnose your own medical problems. And along with the WIMPOS system, Conquer also supplies the MIDI and modem software with the machine. The programming languages bundled with the Chestnut are BASIC and a C compiler.

Despite Conquer's claims during the six months prior to the machine's launch, the Chestnut offers little that's new. Its main selling points are likely to be its low price and bundled features which, for the company's previous machines, were sold as optional extras. But as computer critics have grown in stature to match their counterparts in the worlds of cinema, theatre and literature, it has become all too easy to pan today's new machines, which only seven or eight years ago would have represented the apex of computing achievement.

# LANGUAGE ASSESSMENT

**For many of us, making use of the BASIC built into our micros is sufficient for most of our programming requirements, but certain applications require more appropriate languages. The choice may seem daunting, but a short assessment of your needs and the languages' capabilities will make the task much easier.**

Until recently, the owners of small microcomputers had only BASIC or assembly language available. Now, most of the common languages are available on even the smallest machines, and you only have to go up as far as an IBM PC-compatible or any other MS-DOS machine to have virtually a complete range. Since compilers and interpreters can be quite expensive, and very few people will therefore want to have more than three or four (if that), the problem of choice remains despite increased availability.

Efficiency, in terms of memory used and execution times, can become an important factor, particularly with programs that are interacting with the external environment. In BASIC, for example, some programs would take so long to write and debug that it would be quicker to learn a more appropriate language and write it in that. A program to interrogate and update a file of stock

## The Good, The Bad And The Usage

**Language:** BASIC
**Good points:** Easy to learn and use, widely available. Cheap. Good provision of arithmetic functions. Good string handling
**Bad points:** Most versions do not have a good module facility or control structures. Non-standardised. Execution is often inefficient and slow. Poor file handling. Restricted range of data types and structures. Easy to write bad, unclear code
**Use:** Short, straightforward one-off programs involving arithmetic or string handling

**Language:** LOGO
**Good points:** Sound mathematical basis, easy to learn at a basic level. Turtle graphics. Good list-processing facilities, good provision for modules, data types and structures. Widely and cheaply available
**Bad points:** Programming beyond the initial stage can get very complicated and cryptic. Many incompatible versions. Often inefficient in execution
Graphics, list-processing, learning about mathematics and advanced programming concepts

**Language:** PASCAL
**Good points:** Well-structured. Good range of data types. Fairly standard. Widely available. Easy to produce high-quality code
**Bad points:** Input/output not clearly defined, file handling in particular is bad
**Use:** Learning good programming techniques. General small-scale semi-mathematical programming

items on-line, for instance, would be a simple task in COBOL using indexed files. Written in BASIC, it would have to use sequential files or a hashing algorithm and direct access files that would make it far more complicated than necessary.

Design features that make a language easy to learn and use also tend to create problems when program size increases, where one of the major requirements is the facility to decompose the program into reasonably self-contained modules that can be programmed independently.

Some languages, like COBOL and FORTRAN, are rigidly defined by international committees, making any change to reflect new hardware and types of processing a slow procedure. However, a program written on one machine should only need recompiling, or at most a few minor changes, to run on a different machine.

Languages like PASCAL and C have a *de facto* standard defined by the inventor(s). Most versions of these languages are fairly consistent with the standard but, particularly in languages like PASCAL where input/output is not clearly defined, each implementor is free to make alterations and additions, although programs using these features will not be very portable. Finally, we have languages like BASIC, where there is no standard.

Yet another factor is the type of translator program used. Interpreters tend to be easy to use and better for program development, but slower and less efficient at running the programs. Compilers are more complicated to use but produce a more efficient final product. This is now being reduced, though, by the introduction of incremental compilers and animated debuggers.

Some languages have spawned hardware developments to enable them to run more efficiently. For example, a lot of work has been put into producing a processor that runs FORTH directly; similarly, there exists a processor that runs PASCAL P-code. It may be possible in the near future to produce a hardware-based compiler, which will make the process of using a compiled language much easier.

The major factor remains, however, the type of application the language is being used for. Let's look now at a few languages and list their strong and weak points. Then we can look at a few typical applications and consider what language might be most suitable.

**Language:** FORTH
**Good points:** Efficient execution, extendable by the user. Good structures
**Bad points:** Too low a level for much serious work. Difficult to understand. Many different versions
**Use:** Low-level work in preference to assembler, particularly controlling hardware

**Language:** FORTRAN
**Good points:** Highly standardised. Large libraries of software available. Exceptional range of mathematical and scientific functions
**Bad points:** Old-fashioned. Poor structures. Easy to write cryptic code. Input/output designed around punched-cards. Strict program layout
**Use:** General scientific, engineering and mathematical programming

**Language:** COBOL
**Good points:** Highly standardised, available on a very large range of machines. Fairly easy to read. Excellent file-handling
**Bad points:** Verbose, difficult to learn. Compilers are large and expensive. Badly structured with a poor range of data types
**Use:** Commercial data processing

**Language:** C
**Good points:** Good range of data types and structures. Good modular structure. Provides access to hardware. Produces fast, efficient code
**Bad points:** Easy to produce cryptic code. Level too low for much serious work. Not properly standardised, relying too much on libraries
**Use:** Systems software as a replacement for assembler. Anywhere where maximum efficiency is required

**Language:** LISP
**Good points:** Good list processing, mathematically sound. A lot of software and support available. Widely used on larger machines
**Bad points:** Difficult to learn, understand and use
**Use:** Artificial intelligence and general list-processing

**Language:** PROLOG
**Good points:** Sound mathematical basis. Easy and fun to use. Supposedly closer to human reasoning
**Bad points:** Not a 'pure' relational language — it has some procedural aspects and needs some awkward features, such as the cut, to run efficiently
**Use:** Artificial intelligence, database applications

**Language:** Assembler
**Good points:** Complete control of all aspects of the machine. Potentially, at least, the most efficient
**Bad points:** Completely non-standard. Difficult to learn and use
**Use:** Maximum efficiency, but only if all else fails

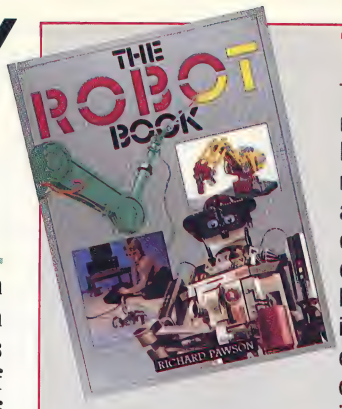
## Applied Linguistics

The table shows a list of applications, followed by the languages most suited, in order of preference:

| Application | Languages |
|---|---|
| Statistical analysis | FORTRAN, BASIC, PASCAL, |
| Stock control | COBOL, PASCAL, BASIC |
| Controlling robot arm | FORTH, C, Assembler |
| Adventure game | PROLOG, C, BASIC |
| Expert system | LISP, PROLOG, LOGO |
| Education | LOGO, PROLOG, PASCAL |

# HARD COPY

**Books about computing have become in recent years one of the boom areas in publishing. It is now possible to find books covering even the most esoteric areas of computing. Here we look at a selection of titles from the diverse range of subjects available, many of which expand on topics covered in the course.**




## The Robot Book

The various aspects of microcomputer control and robotics are informatively covered in The Robot Book in a highly colourful format. The history of robotics, the industrial, educational and domestic applications of robots, how they work, and practical construction projects comprise the four main parts of the book. The first three provide a panoramic background to the subject, while the last section incorporates this knowledge into plans to build your own machines. Lego and Fischertechnik components are used throughout and full instructions are given to interface the finished projects to your micro.

This last section, in particular, augments some of the projects undertaken in our Workshop series, detailing the construction of, among other things, draughts-playing and card-dealing robots, walking devices and static arms. Each is well presented with easy-to-follow diagrams and explanations.

**Title:** The Robot Book
**Author:** Richard Pawson
**Publisher:** Windward
**Price:** £7.95
**ISBN:** 0 7112 0414 4

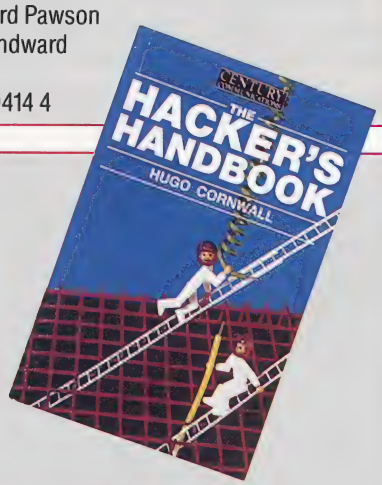
## The Hitch-Hiker's Guide To Artificial Intelligence

The Hitch-Hiker's Guide To Artificial Intelligence is a friendly and informative introduction to AI and develops many of the topics covered in our series on the subject (beginning on page 1221).

Problem solving, natural language and computer creativity are just some of the topics covered in the 10 chapters, which are liberally illustrated with appropriate and instructive BASIC programs. This is one of the book's assets, since you'll need no knowledge of the specialist AI languages.

The book is available for users of the BBC Micro, Apple II, IBM PC and the Amstrad range of micros.

**Title:** The Hitch-Hiker's Guide To Artificial Intelligence
**Authors:** Richard Forsyth and Chris Naylor
**Publisher:** Chapman and Hall/Methuen
**Price:** £8.95
**ISBN:** 0 412 26970 8

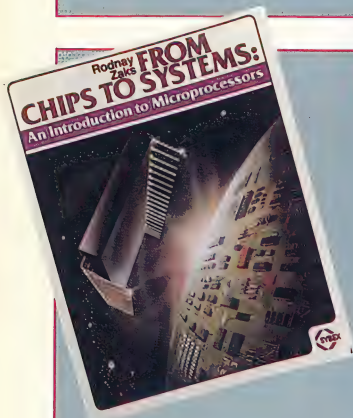



## The Hacker's Handbook

The Hacker's Handbook grabbed the national headlines when the police, fearing the disclosure of confidential information, insisted on reading the book prior to its publication. But it tells experienced hackers little they don't already know, and gives only vague directions to 'the novice'.

The book amounts to a detailed account of communications devices, how they are implemented on a variety of systems, and what equipment is needed to get you started. Viewed in this light, The Hacker's Handbook is an invaluable guide to data networks, from bulletin boards to the large public databases, and can thus be recommended for anyone interested in using a modem — even if it's not going to be for breaking into NATO computers.

**Title:** The Hacker's Handbook
**Author:** Hugo Cornwall
**Publisher:** Century
**Price:** £4.95
**ISBN:** 0 7126 0650 5

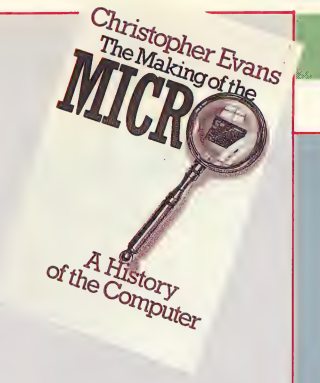



## From Chips To Systems

From Chips To Systems has collected within its covers all the major aspects of computer hardware, removing much of the mystery that prevents many of us from delving into the subject. Assuming only a limited knowledge of computing and electronics, Rodnay Zaks outlines the workings of the individual components, and details how they interconnect to form complete systems. Taking us from the principles of microprocessor design to interfacing, Zaks then looks at simple microprocessor applications, such as a cassette motor controller, an analogue-to-digital converter and a microwave oven controller.

**Title:** From Chips To Systems
**Author:** Rodnay Zaks
**Publisher:** Sybex
**Price:** £12.95
**ISBN:** 0 89588 063 6

## The Making Of The Micro

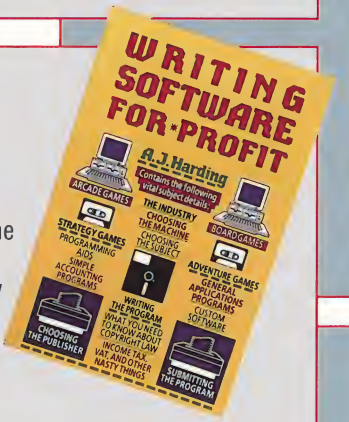



Like The Soul Of A New Machine by Tracey Kidder (see page 940), The Making Of The Micro provides some fascinating background information about the development of computers, in this case from the historical perspective. The works of figures like Babbage, von Neumann and Pascal are discussed in detail, and some of them are brought more to life with accounts of their personal affairs.

One point this book implicitly stresses is just how old computers really are, and how we have come to take them for granted without giving due credit to their pioneers. Few people will know, for example, that Vannevar Bush made a great advancement by replacing the mechanical elements in his 'differential analyser' with thermionic tubes. This is an interesting, though brief, insight into the background of the new technology.

**Title:** The Making Of The Micro
**Author:** Christopher Evans
**Publisher:** Oxford University Press
**Price:** £1.95
**ISBN:** 0 19 286035 6

## Writing Software For Profit

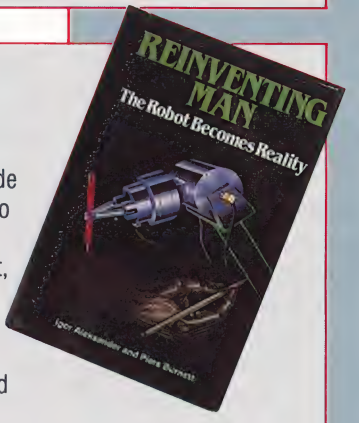



Innumerable books have been written that cater to would-be programmers who dream of writing a piece of software that will make them a fortune. Writing Software For Profit, however, is intended to point out the pitfalls of this quest, encouraging some consideration before writing your first line of code.

In the process, A J Harding provides an overview of the software industry, illustrating the kind of programming and presentation expected — an aspect of the business often overlooked by other publications dealing with this topic.

**Title:** Writing Software For Profit
**Author:** A J Harding
**Publisher:** Virgin Books
**Price:** £4.95
**ISBN:** 0 86369 0572

## Reinventing Man

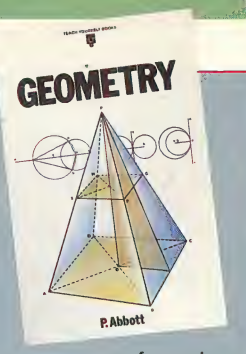



Reinventing Man has become something of a standard reference work for those interested in the field of robotics. Its format, though, is not to provide a wealth of technical detail on robots. It is, rather, to look at the history of the 'mechanical man', contrasting this with actual and likely development, and debunking several myths in the process.

The authors explain in detail how current technology can lead to the hardware and software likely to be needed to develop the robots so beloved by science fiction aficionados, and predict the limitations this technology is likely to have. Anyone interested in artificial intelligence and robotics will find this book absorbing, both for its wealth of information and its fascinating subject matter.

**Title:** Reinventing Man
**Authors:** Igor Aleksander and Piers Burnett
**Publishers:** Kogan Page
**Price:** £9.95
**ISBN:** 0 85038 741 8

## Teach Yourself Geometry

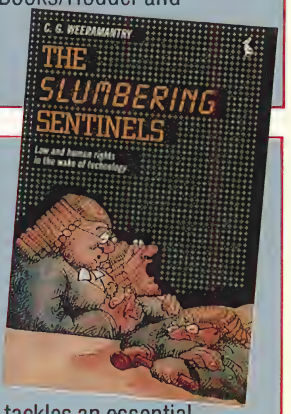



One of the difficulties many programmers face when they begin to tackle more ambitious projects is a lack of mathematical or algebraic theory, knowledge of which can be especially useful in devising new, more compact coding methods.

Teach Yourself Geometry, like the other books in the range, gives a thorough grounding in its subject with a number of test exercises, the answers to which are provided at the end of the book. It's particularly useful if you do a lot of graphics programming, and is typical of a whole range of literature which, while not specifically directed at computer users, can nevertheless be extremely useful.

**Title:** Teach Yourself Geometry
**Author:** P Abbott
**Publisher:** Teach Yourself Books/Hodder and Stoughton
**Price:** £4.50
**ISBN:** 0 340 26164 1

## The Slumbering Sentinels



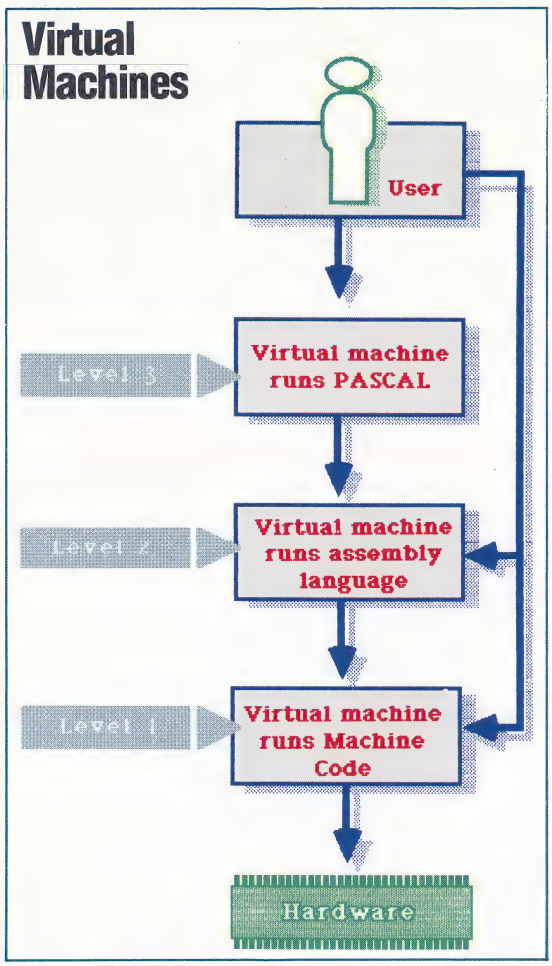

The Slumbering Sentinels tackles an essential aspect of the emerging new technology – how it is approached, encompassed and dealt with by the law. It also focuses on moral dilemnas posed by technological development and its effect on the rights of the individual.

The book is not exclusively about computers, but it does touch on the subject with distressing frequency, illustrating the inadequacy of our present constitutional framework in dealing with problems such as the storage and disclosure of sensitive information, displaying its unwillingness to tackle them – or so it might appear. The legal aspect of computing is rarely discussed, and that makes this book an interesting study for anyone involved in the use — particularly the commercial use — of new technology.

**Title:** The Slumbering Sentinels
**Author:** C G Weeramantry
**Publisher:** Penguin (Pelican Originals)
**Price:** £3.50
**ISBN:** 0 14 022498 X

# IN TRANSLATION

**Level Of Involvement**
From the programmer's point of view, a computer may be regarded as three 'virtual machines'. At level 3, the computer in the example runs PASCAL. At level 2 it runs assembler, and at level 1 machine code drives the hardware directly




CAROLINE CLAYTON

**Our final instalment on programming the 68000 considers the operation of the assembler. We start from a high-level view of where the assembler fits into the overall programming process, and then consider its features in detail. Our discussion covers more than just the instruction statements, since we will need to look at the assembler as a 'tool' for producing low-level software.**

In previous instalments, we have repeatedly used the assembler level instructions to illustrate the features of the 68000 microprocessor. In our discussions, we have assumed that the computer actually does obey these instructions as they appear in their assembler form. In fact, of course, the machine interprets a binary code that represents the higher level assembler instructions, and it is the job of the assembler to *translate* these instructions into machine code.

However, it is convenient to consider the execution of our programs as though the machine actually does execute those programs at source level. This level can be very high — for example, application packages like WordStar — or at a slightly lower level — such as PASCAL programs — or even lower still at the level of the assembler programs. This gives us a multi-level approach to our view of the machine, which consists of discrete layers or levels that become *virtual* machines, executing a given programming language, as shown in the diagram.

The way a program runs at any level is either by:

- **Translation:** A program at one level is translated into the form of the lower level machine. For example, we can translate a PASCAL program at level 3 into assembler to run at level 2 with a compiler.
- **Interpretation:** A program at one level is interpreted by a program (called an 'interpreter', of course) running on a lower level machine. For example, the machine interprets the binary bit patterns representing the instructions into machine (or register level) operations. Equally, level 2 could also interpret PASCAL programs from level 3.

It is the process of translation that most concerns us here. There are many forms of translation, ranging from compilers to assemblers, but they all have one thing in common — they all take high level instructions and produce the corresponding lower level instructions. The original source instructions are then no longer required, and are in a sense 'thrown away' by the translator (an interpreter would, of course, still require the original source program).

The object of the assembler is to translate the high level assembler instructions into the binary code that the machine can execute. For example:

```
MOVE.W D3,D5
```

would be translated into:

```
0011 101 000 000 011
```

The 0011 corresponds to 'move a word'; 101 000 corresponds to 'to D5'; and 000 011 corresponds to 'from D3'.

However, working at the level of coding bits is far too tedious and prone to error, so at the very least we need mnemonics to enable us to remember instructions and data objects. For example, MOVE.W TOTAL,D4 means that we can refer to locations by symbolic names that indicate the meaning of the contents of that location (TOTAL in this example).

The sorts of errors that would be encountered if we coded bits by hand include:

- Getting instruction codes (op-codes) wrong.
- Incorrect absolute addressing.
- Assigning the wrong number of bytes per instruction.

Thus, manual translation for anything more than, say, half a page of instructions is clearly out of the question. Hence the role of the assembler in translating the source code into object code.

## THE ASSEMBLY PROCESS

The assembly process starts with the assembler reading the source statements from the source text file (written in ASCII text) and producing a listing file (or printing it) and an object file (or loading it into the machine). Our Assembly Listing Format file, which shows the components of a translation, is a simple program that performs an arithmetic calculation on the elements of an array.

Several points are worth making about the listing. First of all, you set a list of all the locations used (LOC on the title line), then a listing of the location contents and then the rest of the line follows with the original source code following an inserted statement number. Any errors reported would be with reference to a statement number. For example, if line 14 contained an error, then E would be printed on that line and a reference made to that line in the error reporting section of the listing.

The format of the binary file is interesting from several points of view. First of all, the file must contain the binary information in a coded form, as well as loader information so that the binary can be correctly loaded into memory. The way this is done is to store the address and contents information in hex-coded binary, and hence the loader must know the binary format. The exact format depends on the assembler, but loader compatability is essential.

The final section of the listing is the symbol table print-out. This table gives the numerical values assigned to the labels declared in the program. For example, INPUT — defined in statement 25 — has a value of 1024 (hex) and is referenced in statement 10. All fairly obvious detail for the relatively small example program above, but for any large program this becomes essential debugging information.

The destination of the output depends on where the assembler is running (the assembler is, of course, a program like any other). If, for example, we use a 'cross-assembler' method, then the assembler runs on a host system like Unix, and the listing and binary information would be in files. The binary file would have to be down-loaded to the target 68000 before running the program.

An alternative to this would be to run the assembler on the target machine, if it had at least some sort of rudimentary filing system. The binary codes are sometimes loaded directly into memory with target assembly and the listings may be printed out directly if a printer is attached.

## Assembly Listing Format

```
LOC      OBJECT                    STMT          SOURCE STATEMENT
                                   1   *This program takes each element of an array and converts
                                   2   *each element to
                                   3   *                INPUT[I]:=2*INPUT[I]+3
                                   4
                                   5
001000                             6         ORG $1000
         =000C                     7   length:    equ 12
                                   8
001000   700C                      9   start:     moveq     #length,d0
001002   41F8 1024                 10             lea       input,a0
001006   4244                      11             clr       d4
001008   4243                      12             clr       d3
                                   13
00100A   3610                      14  loop:      move.w    (a0),d3
00100C   C7FC 0002                 15             muls      #2,d3
001010   D67C 0003                 16             add.w     #3,d3
001014   30C3                      17             move      d3,(a0)+
001016   D843                      18             add       d3,d4
                                   19
001018   5340                      20             subq      #1,d0
00101A   6600 FFEE                 21             bne       loop
00101E   31C4 103C                 22             move      d4,sum
001022   4F40                      23             trap      #0
                                   24
001024   0001 0002 0003 0004       25  input:     dc.w      1,2,3,4,5
         0005
00102F   0006 0007 0008 0009       26             dc.w      6,7,8,9,10
         000A
001038   000B 000C                 27             dc.w      11,12
                                   28
00103C   0000                      29  sum:       dc.w      0
                                   30
                                   31             end

No errors found in this assembly

         SYMBOL                    DEFN          VALUE     REFERENCES
INPUT                              25            1024      10
LENGTH                             7             000C      9
LOOP                               14            100A      21
START                              9U            1000
SUM                                29            103C      22
```

## THE ASSEMBLER MECHANISMS

The mechanisms involved in the assembler give us an insight into the problems that may occur (as they inevitably will do!) Usually, the assembly process is a 'two pass' arrangement (see page 1875). On the first pass, the assembler will:

- Decode the assembly mnemonics (MOVE, ADD, MULS and so on).
- Count the address bytes (so that we can work out the addresses).
- Construct a symbol table (which tells us the values allocated to each symbol).
- Output any syntax errors (for example, ADDQQ is an illegal instruction).

When the assembler reads statement 9 on the first pass through our Assembly Listing Format program, the label contents of that location will contain 41F8. This corresponds to a 'quick move' to D0 in immediate mode with a constant of 12 (from LENGTH in the symbol table). For statement 10, the assembler can set location 1002 to 41F8 for the LEA instruction, but it cannot yet set the address of INPUT. So, location 1004 is left blank until the second pass when the address of INPUT is known.

On the second pass, the assembler can substitute the now known addresses wherever

they are used in the program, and output all the binary code and program listing. There may also be errors to clear up too, but these are usually taken care of by the first pass.

We can use the assembler as a convenient calculator too, by creating symbolic expressions that evaluate to the value of an operand. For example, these introductory lines of an assembly listing:

```
LENGTH   DC.W      (START-ARRAY)*4
ARRAY    DC.W      1,2,3,4,5
START    MOVE.W    LENGTH,D3
```

Here we need to compute the array length in bytes (START-ARRAY multiplied by four), and use this in the program (to set data register 3 in this case). If at some time later we change the array length in the assembler program then LENGTH will be reset automatically.

## DOCUMENTATION AIDS

Another aid the assembler provides is in the documentation areas. We can add comments to our programs that tell readers about the meanings of instructions, and we can also include details of, for example, register conventions and program design. Consider the following listing:

```
*Program to input a string and verify it
*Text is input via the ACIA and verified
*A. Person — Dec 85
*
*Register convention
*
*Address registers
*       A1 pointer to input string
*       A2 pointer to stored keyword
*       A3 pointer to current character
*Data registers
*       D0 input character
*       D1 output character
*
START  JSR  INIT        initialise the I/O
       JSR  READSTRING
                        read the input string
       JSR  VALIDATE    and check it
       JSR  SIGNAL      signal if OK
       BRA  START       loop for ever
```

Reading these comments tells us about the program, its structure and register conventions. Probably, it would be asking too much for more details of the program at this level, but of course, the reader can go down another level and look at the comments in the subroutine code if any more information is required. The important point is that the program has been reasonably well documented with the use of comments.

## ASSEMBLER DIRECTIVES

The assembler also allows us other programming aids in the general area of *assembler directives*. These can be pure documentation aids like the TTL or 'title directive', which instructs a given title to be printed on each page of the listing, or a means of passing information to the assembler. For example, we may wish to signal the end of our input text to the assembler with the END directive. Or we may wish the assembler to start assembling at a specific address by using the ORG instruction. So our program module would certainly contain the following lines:

```
*The documentation comments
*With our name and a date at least!
       TTL This is my program
       ORG $1000       *an origin instruction
START  MOVE D1,D2      *and some instructions
                       *lots more instructions
       END             *an end directive
```

The more important assembler directives are summarised in the table.

### Assembler Directives Table

| | |
|---|---|
| ORG | Short for 'origin'. Specifies start address of code |
| RORG | As for ORG, but for PC relative code |
| TTL | Sets a title on every listing page |
| END | No more text to assemble |
| EQU | 'Equate' — assigns a value to the symbol specified (e.g. LENGTH EQU 10+50*ARRAY1 — so if ARRAY1 is 12 then LENGTH becomes 610) |
| DC | 'Declare constant' — assigns a constant value (byte, word or long word length) e.g. PARAM DC.W 12, 23 sets PARAM as a word containing 12 and PARAM+2 as a word containing 23 |
| DS | Reserves an area of uninitialised data (e.g. STACK DS.B 100 sets an address space of 100 bytes called STACK) |
| LLEN | Sets line length |
| PAGE | Issues form-feed during listing |

## ASSEMBLER FORMAT

Naturally, assemblers produced from different manufacturers will differ in detail as far as their operation and format are concerned. Nevertheless, most assemblers conform to a source layout that is similar to the following layout. First of all, comments can be declared if a * is used as the first character in a line, or after at least one space following an instruction. So both of the following are legal:

```
*This is a comment
START ADD D1,D2 *and so is this
```

What constitutes an instruction has, of course, to be defined, and we can say that this is made up of three optional fields:

- **Label field:** Each name used as a label must start with an alpha character and be less than 30 alphanumeric characters in total length. Note that if this field is omitted then at least one space must be inserted.
- **Op-code field:** One of the 68000 set of instructions must be used.
- **Operand field:** Legal instruction operands must be used following at least one space after the op-code field. There must be no spaces in the operand field even if two operands are used.

Therefore, examples of illegal assembler statements would include:

```
This is not a comment because I've left out the *
MOVE    D1,D2       *don't leave a space
                    *between the two operands
MOVE    8TOIT,D5    *names must start with a
                    *letter AND there must be
                    *at least one space
                    *following the operands
RTS                 *that's ok
RTS     D1          *but that's not (D1 not
                    *allowed)
*end of erroneous example with correct comment!
```

Finally, a word about alphanumeric representation. Numbers are represented to the assembler as decimal values unless a $ sign precedes the number. So, for example, 1234 is decimal and $1234 is hex. For ASCII literals, the characters are enclosed in quotes. For example:

```
'This is the end of the course!'
```

We have discussed here the use of the assembler as a programming tool, not merely as a means of getting code into a 68000. The facilities offered are quite good especially with the macro feature and reasonably good assembler directives and error messages. The documentation aids have also been highlighted, since it should never be forgotten that other people are *certain* to read your listing!

**A Slower Pace Of Life**
The QL possesses a 68008 microprocessor, which is identical to the 68000 except for its slimmed down 8-bit data bus. The two chips are entirely instruction compatible and therefore our series on programming the 68000 is also applicable to programming the 68008. The only difference you will find will be one of speed!

## MACRO Facility

A powerful programming tool for the development of large programs, the MACRO facility is a method of text substitution in a source program. Where a 'macro' is specified, the previously defined macro text will be output. An ERROR macro, for example, might be defined as follows:

```
ERROR   MACRO
        MOVE.B      #3, ERRORLOC
        JSR         SIGNAL
        ENDMACRO
```

Thereafter the word ERROR would produce — at assembly time — the following series of instructions:

```
MOVE.B  #3,ERROR.LOC
JSR     SIGNAL
```

We can expand this facility to include the use of parameters at every call if we require this. For example:

```
                TTL Macro Example
                ORG         $1000

ADDON   MACRO
        ADD         \1,D2
        ADD         D2,\2
        ENDM

        ADDON       AVAL,SUM
        ADDON       BVAL,SUM
        ADDON       SUM,TOTAL
AVAL    DC.W        0
BVAL    DC.W        0
SUM     DC.W        0
TOTAL   DC.W        0
        END
```

In this example, the \1 and \2 refer to the first and second parameters. This would produce, at assembly time:

```
3               ORG         $1000
4
5   ADDON       MACRO
6               ADD         \1,D2
7               ADD         D2,\2
8               ENDM
9
10              ADDON       AVAL.SUM
11+             ADD         AVAL,D2
12+             ADD         D2,SUM
13              ADDON       BVAL,SUM
14+             ADD         BVAL,D2
15+             ADD         D2,SUM
16              ADDON       SUM,TOTAL
17+             ADD         SUM,D2
18+             ADD         D2,TOTAL
19
20  AVAL        DC.W        0
21  BVAL        DC.W        0
22  SUM         DC.W        0
23  TOTAL       DC.W        0
24              END
```

Note that the + signs after the statement numbers indicate the lines inserted by the macro processor

# SMOOTHING THE TRANSITION

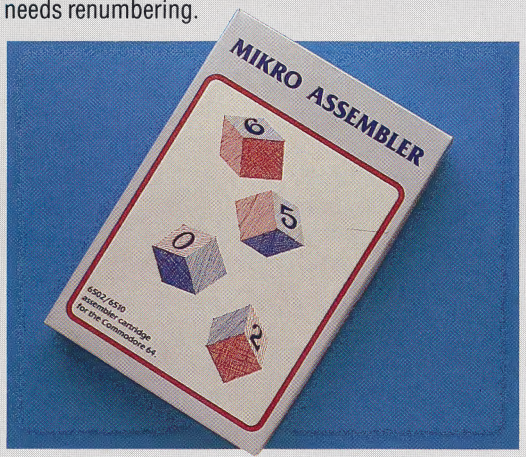
**In the previous instalment, we looked at some assembler packages for the BBC Micro, Sinclair Spectrum and the Amstrad range. We conclude this survey by examining Supersoft's Mikro, a cartridge-based assembler/monitor for the Commodore 64. The program's use of BASIC format source programs has made it very popular with novice users.**

Among the problems you will encounter when first writing machine code programs are the lack of 'protection' from the machine inherent in machine code programming, and the unfamiliar method of preparing running programs. The former is the price you pay for the power that machine code programming can give you.

Supersoft's Mikro assembler cartridge allows source program text to be entered using a full screen editor and line numbers, as used by Commodore 64 BASIC. Because most people learning machine code on their micro will have first mastered BASIC, this method of entry will be familiar and help to smooth the transition from BASIC programming to working in machine code.

## Supersoft Mikro

Supersoft's Mikro is the long-running standard for the Commodore 64. Its advantage is that it's housed in a plug-in cartridge, providing a full assembler and monitor neatly integrated with the Commodore BASIC. Source programs are entered as normal BASIC program lines; you can't RUN the program, of course, but you use the normal SAVE and LOAD commands and the BASIC screen editor to store and correct your source code. Although this is a crude solution, the Mikro adds an AUTOmatic line number command, a DELETE lines command and a FIND string to make editing easier. The essential RENUMBER command is missing and you will need to load a separate utility if your program needs renumbering.




LIZ HEANEY

During assembly, the Mikro can produce a full assembly listing on a printer, but not on screen, and can produce a sorted list of symbols used in the programs. Assembler directives are adequate, the most useful of which is LNK, which allows a program to be written in sections and then assembled from cassette or disk as one program. This facilitates development of programs of up to around 12 Kbytes of object code.

The Mikro has a machine code monitor similar to many available for the Commodore 64, and to those built into the Commodore PET range. Again, this is adequate since it does not include breakpoints or tracing. A short 16-page manual is included, and although it describes the commands, it's not as detailed as it might be. In summary, the Mikro is adequate and reliable but may prove cumbersome for developing large programs.

**Assembler Type**
Two pass, standard mnemonics

**Limits**
Source Code: 30K, but can link files
Symbol Table Size: 11K

**Assembler Directives**

| | |
|---|---|
| * | Used instead of ORG and DEFS. It refers to the current location, and an instruction such as *=$6000 will instruct the assembler to start the program at location $6000. An instruction such as *=*+256 would reserve a 256 byte space for data within the program |
| = | Assign a value to a symbol |
| WOR | Store 16-bit value(s) |
| BYT | Store 8-bit value(s) |
| TXT | Store ASCII string |
| LNK | Continue assembling from named source file |
| OUT | Display assembly listing on printer |
| OFF | Turn off listing |

**Assembly Options**
None

**Arithmetic**
+, =, < and > (high and low bytes of 16-bit values). Decimal, hex and binary.

**Price**
£57.50

Supersoft (01 861 1166)

# NOW IS THE TIME TO ORDER ANY COPIES OF THE HOME COMPUTER ADVANCED COURSE YOU ARE MISSING.

COPIES OF ANY WEEKLY ISSUES CAN BE OBTAINED, SUBJECT TO THE AVAILABILITY OF STOCKS, BY USING THIS REPLY PAID ORDER FORM AND MARKING CLEARLY WHICH ISSUES YOU REQUIRE TO BE SENT TO YOU. ISSUES 1-53 ARE 80 PENCE EACH, ISSUES 54-96 ARE 90 PENCE EACH. PRICES ARE INCLUSIVE OF POSTAGE AND PACKING.
PLEASE ENCLOSE YOUR CHEQUE/POSTAL ORDER MADE PAYABLE TO ORBIS PUBLISHING LIMITED.

FOLD 2

## BACK NUMBERS ORDER FORM.

PLEASE SEND ME THE BACK NUMBERS OF THE HOME COMPUTER ADVANCE COURSE I HAVE CIRCLED BELOW:
EACH ISSUE COSTS 80 PENCE INCLUDING POSTAGE AND PACKING.

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 |
| 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 |
| 49 | 50 | 51 | 52 | 53 | | | | | | | |

FOLD 3

EACH ISSUE COSTS 90 PENCE INCLUDING POSTAGE AND PACKING.

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 |
| 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 |
| 78 | 79 | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 |
| 90 | 91 | 92 | 93 | 94 | 95 | 96 | | | | | |

## BINDER ORDER FORM.

PLEASE SEND ME THE BINDER VOLUME NUMBERS I HAVE TICKED BELOW:
☐ VOLUME 1 ☐ VOLUME 2 ☐ VOLUME 3 ☐ VOLUME 4
☐ VOLUME 5 ☐ VOLUME 6 ☐ VOLUME 7 ☐ VOLUME 8

BINDERS ARE £3.95 EACH. BINDERS MAY BE SUBJECT TO IMPORT DUTY AND/OR LOCAL TAXES.
N.B. READERS NOT IN UK OR THE REPUBLIC OF IRELAND, SEE INSIDE FRONT COVER FOR DETAILS OF HOW TO OBTAIN BINDERS AND BACK NUMBERS.

FOLD 4

FOLD 4

I enclose a cheque/postal order made payable to: Orbis Publishing Ltd. for a total of £____which I understand includes the cost of postage and packing.

NB: Please allow 28 days for the delivery of your binders.

When you have completed the order form fill in your name and address in the space provided.

Then cut along the dotted line to detach the page, enclose your cheque/postal order, and fold the page carefully – following the instructions to complete the reply paid envelope.

NO STAMP NECESSARY.

FOLD 1

IF YOU ALREADY HAVE AN ACCOUNT NO. FOR YOUR BINDERS PLEASE FILL IT IN HERE.

Complete the section below with one letter or figure per space.

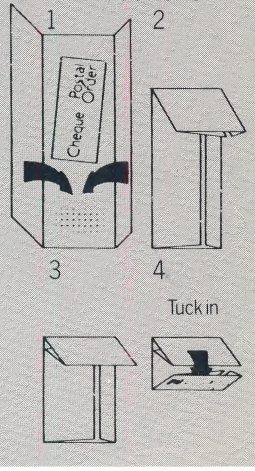
| | | | |
|---|---|---|---|
| MR | | INITIALS | SURNAME |
| MRS | | | |
| MISS | | | |

ADDRESS & POST CODE

TELEPHONE NO. INCLUDING STD CODE OR EXCHANGE NAME

FOLD 1

FOLD 4
FOLDING INSTRUCTIONS

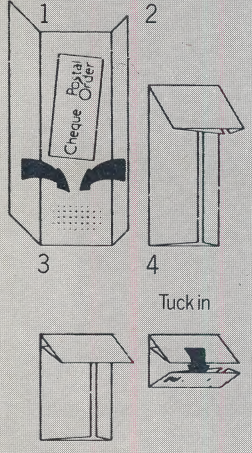


# IT'S TIME TO COMPLETE YOUR COLLECTION OF THE HOME COMPUTER ADVANCED COURSE.

# FOR YOUR *LAST* OPPORTUNITY TO ORDER ANY BACK ISSUES OR BINDERS YOU MAY HAVE MISSED, PLEASE SEE THE ORDER-FORM OVERLEAF.

FOLD 4

FOLDING INSTRUCTIONS




FOLD 1

Postage will be paid by licensee

Do not affix Postage Stamps if posted in Gt. Britain, Channel Islands or N. Ireland.

BUSINESS REPLY SERVICE
Licence No. SW4035.

The Home Computer
Advanced Course Binders
Orbis House, 20-22 Bedfordbury
London WC2N 4BR

2

FOLD 1

GUARANTEE GUARANTEE GUARANTEE GUARANTEE GUARANTEE

**GUARANTEE**

**If you are not entirely satisfied with your binder, send it back immediately and it will be either exchanged, or, if you prefer, your money will be refunded in full.**

**GUARANTEE**

GUARANTEE GUARANTEE GUARANTEE GUARANTEE GUARANTEE